Juan David Arango

Juan Sebastián Ramírez

**Presentacion Reto Fianl**

**Parte A:**

* Se utilizo el TAD tablaDeSimbolos, específicamente se utilizó la estructura TablaHashLinearProbing, la cual era la tabla que guardaba las compañías. Recibía como key el nombre de la compañía y como value guardaba una compañía de clase companiaTaxis, la cual a su vez contenía una TablaHashLinearProbing en la que se guardaban los taxis de esa compañía, las llaves eran los id de los taxis y los values eran un taxi de clase Taxi.
* Elegimos esta estructura de datos debido a la facilidad que tiene para insertar, pero sobre todo para encontrar elementos. Sabíamos que sería más efectivo si podíamos acceder fácilmente a cualquiera de las compañías para poder revisar sus datos y también esperábamos que fuera sencillo poder insertar nueva información de las grandes bases de datos que íbamos a manejar.
* Depende de la acción. El tiempo de carga de la lista de compañías debería de ser de aproximadamente O(n), siendo n la cantidad de viajes. La teoría es que debería ser O(1) debido a que se están usando tablas hash, no obstante, cuando se cargan los archivos se puede evidenciar fácilmente que la velocidad de carga se ve afectada según la cantidad de datos. Es por esto por lo que se puede decir que no es constante. En cuanto a la carga de los tops de las compañías con más taxis, su complejidad estimada debería ser de O(n log(n)), siendo n la cantidad de compañías totales, debido a que está utilizando un Quicksort para ordenarlas.

**Parte B:**

* Para esta parte son importantes 3 clases, la clase CompaniaTaxi, la clase Taxi y una exclusiva de la parte B: la clase Servicio. Al igual que la parte A, la información se cargó en una tabla de símbolos de estructura hash table que maneja las colisiones por medio del Linear Probing. El modelo tiene una tabla de compañías de taxi, cada compañía tiene una tabla de Taxis, y cada taxi tiene una lista simplemente encadenada de servicios.
* Para el control de los servicios de cada taxi utilizamos una Lista encadenada para intentar disminuir el uso de memoria al máximo, y debido a que asumimos que la lista de servicios que cada taxi realizo no sería tan grande era una opción viable y fácil de implementar.
* La complejidad temporal del algoritmo podemos definirla como:

**.**

Donde Calculo Puntos es el algoritmo utilizado para calcular los puntos de cada taxi, donde su complejidad es de O(n), pues recorre todos los datos**;** y Sort es la complejidad del algoritmo utilizado para ordenar los datos una vez calculados los puntos, en este caso como se emplea un Shell Sort es difícil dar una complejidad exacta, pero asumiendo un caso “promedio”, tendría una complejidad de O(n3/2), por lo tanto, la complejidad del requerimiento es aproximadamente

**Parte C:**

* Se utilizo el TAD iDiGraph, específicamente se utilizó la estructura DiGraph, la cual era la tabla que guardaba las compañías.
* Elegimos este TAD debido al reto en sí, requeríamos que un elemento pudiese guardar múltiples conexiones a otros elementos, en este caso las comunity áreas y las conexiones se podían representar como los viajes y sus informaciones respectivamente, esta información era la parte más importante de este punto, ya que era la que permitía que se pudiera encontrar la mejor hora para viajar entre una área y la otra.
* La complejidad de un grafo no es tan fácil de comprender, sin embargo, estimaríamos una complejidad de O(n), siendo n la cantidad de viajes, nos gustaría pensar que tiene más que ver con la cantidad de comunity áreas, sin embargo, se puede notar que la carga se ve afectada según el tamaño del archivo que se utilice y estos archivos lo que principalmente contienen son viajes, realmente los comunity áreas no son tantos como la enorme cantidad de viajes que hay, es por esto que se puede notar que se está afectando es por los viajes. La complejidad de un grafo es difícil de calcular, sobre todo porque depende de muchos factores, como por ejemplo, como están compuestos sus vértices o cómo están compuestos sus arcos. Es por esto que hacemos este estimativo de O(n).